Performance Overheads of Confidential Virtual Machines

Mingjie Yan and Kartik Gopalan
Computer Science Department, Binghamton University
Binghamton, NY, USA - 13902-6000
{myan28,kartik}@binghamton.edu

***Abstract*—A Confidential Virtual Machine (CVM) is a virtual machine (VM) whose memory is encrypted using trusted hardware support to prevent unauthorized access to its contents, including by the hypervisor. AMD Secure Encrypted Virtualization (SEV) provides hardware support for CVMs on AMD processors and has been used by several cloud operators to provide trusted execution environments to cloud users. In this paper, we examine the performance overheads of CVMs across three generations of AMD SEV using a number of CPU, memory, and I/O benchmarks. Our findings indicate that CPU-intensive workloads running on a CVM do not experience significant performance difference compared to a non-confidential VM. However, we observe that some workloads that are sensitive to cache/memory latency may experience a performance drop of up to 2.5%. Pure memory-intensive workloads are observed to experience up to 4.3% overhead. Disk I/O from CVMs experiences a significant performance impact when using SEV, with up to a 56% performance penalty. Network I/O, on the other hand, experiences up to a 36% overhead. Workloads with a mix of memory and I/O accesses experience an overhead of up to 14%. Our work complements and extends the existing understanding of the performance of this important and rapidly evolving technology.**

***Index Terms*—Confidential computing, virtual machines**

I. Introduction

Confidential computing aims to protect sensitive customer data, such as financial transactions, medical records, and intellectual property, from unauthorized access on third-party cloud platforms. An emerging approach to confidential computing is to use processor-level hardware support to encrypt the memory of a Confidential Virtual Machine (CVM), making it impossible for the hypervisor or any host software to access the CVM's memory contents. While storage encryption can protect persistent data that is not in active use by an application, hardware-based CVM encrypts data loaded in memory that is actively in use, protecting it from other co-located VMs and even from a malicious host. Some prominent technologies that support CVMs include AMD Secure Encrypted Virtualization (SEV) [1]–[3] and Intel Trust Domain Extensions (TDX) [4].

A CVM can strengthen confidentiality guarantees, but it can also introduce significant overheads, because all data in a CVM's protected memory is encrypted by the memory controller when written and decrypted when read. While this, on its own, adds additional latency to memory accesses, more serious performance impacts may arise from the need to communicate some data with the hypervisor for I/O operations.

In this paper, we present a performance study of CVMs that use the AMD SEV hardware, which is available on select AMD processors and is supported by major hypervisors. At the time of writing this paper, Intel TDX is not yet widely available, and hence not considered.

AMD SEV works by generating a unique encryption key for each CVM. This key is stored in an AMD Secure Processor (AMD SP), a separate security co-processor based on ARM Cortex A5, that is isolated from the main CPU. When a CVM is started, AMD SP generates and stores unique keys to encrypt all of the CVM's protected memory. A high performance Advanced Encryption Standard (AES) engine in the memory controller performs inline encryption/decryption of data in protected memory. This encryption is transparent to the CVM, which runs unmodified OS and applications.

We investigate the performance of CVMs over three generations of AMD SEV VMs compared to General (unencrypted) VMs, from various aspects including CPU, memory, I/O, and real-world application performance. While there have been other recent efforts [5]–[9] to evaluate the performance of AMD SEV, they focused on either the first generation SEV support or only on specific types of workloads. Our work complements these prior works to provide a comprehensive and more up-to-date understanding of this important and rapidly evolving technology. Our key findings are as follows.

- CPU-intensive workloads in a CVM do not experience significant performance difference compared to a General VM. However, workloads sensitive to cache/memory latency may experience up to 2.5% performance loss.
- Memory-intensive workloads, such as the STREAM benchmark, experience up to 4.5% overhead.
- Disk I/O from CVMs experiences the worst performance impact, with up to 56% overhead, while network I/O experiences up to 36% overhead.
- Mixed memory and I/O workloads, such as Redis benchmark, experience an overheads of up to 14%.

Next, we first provide a background of three generations of AMD SEV hardware. Then we present the evaluation and analysis of AMD SEV with CPU, memory, and I/O intensive benchmarks, followed by related work and conclusions.

II. AMD SEV Background

The AMD-V architecture [10], [11] provides hardware support for VMs on AMD processors. AMD SEV extends

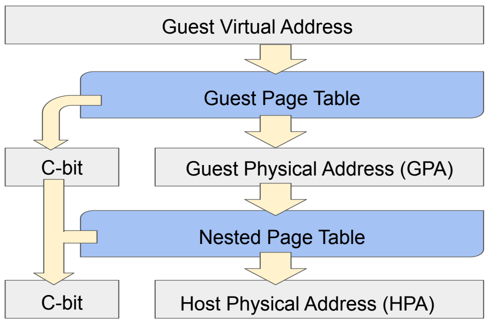

Fig. 1. Address Translation in AMD SEV. Guest pages with C-bit set to 1 in their guest page table entry are encrypted by AMD SEV hardware.

AMD-V to provide an additional layer of strong cryptographic isolation between VMs and the hypervisor. Here we provide background on three generations of the AMD SEV technology.

A. First Generation SEV

The first generation SEV [1] tags code and data of a CVM using a VM ASID (Address Space Identifier). The ASID is used by the CPU to identify the VM to which the data belongs and allows access only to the authorized VM. Additionally, the AES engine encrypts the data outside the CPU with a 128-bit key. The SEV firmware in the AMD SP manages all keys securely to prevent access by the untrusted host, besides performing authentication and attestation. The AMD SP itself is an ARM Cortex-A5 running with ARM TrustZone extension [12]. Figure 1 shows the memory encryption process. When SEV is enabled, the CVM's guest OS boot code sets a C-bit (as in enCryption bit) in the guest page table entries for pages that must be protected. The memory controller scans the C-bit, the AMD SP provides the encryption key based on each ASID, and the AES Engine performs the encryption. Each encrypted memory region is pre-reserved.

B. SEV with Encrypted State (SEV-ES)

The second generation SEV-ES [2] protects guest VMs from register state attacks by a malicious hypervisor. Such attacks could involve reading guest register values, writing malicious values, or replaying old state back into the VM, leading to data leaks, control flow modification, and unintended behaviors in the guest OS. To counter these attacks, SEV-ES encrypts the guest register state that only the guest is permitted to modify.

The Virtual Machine Control Block (VMCB) is a data structure utilized by a hypervisor to manage and control a VM. It contains information about a VM's virtual CPU (VCPU) execution states, including its registers, memory mappings, and other information. When a VM attempts a privileged operation that a hypervisor must emulate then a VM Exit occurs, meaning that the control of the CPU is transferred from the VM to the hypervisor. To handle a VM Exit, the hypervisor uses the VMCB to save/restore the VCPU state.

In General VMs, the VMCB directly stores the VCPU register state. When SEV-ES is enabled, the VMCB is divided into two sections: the "control area" and the "save area."

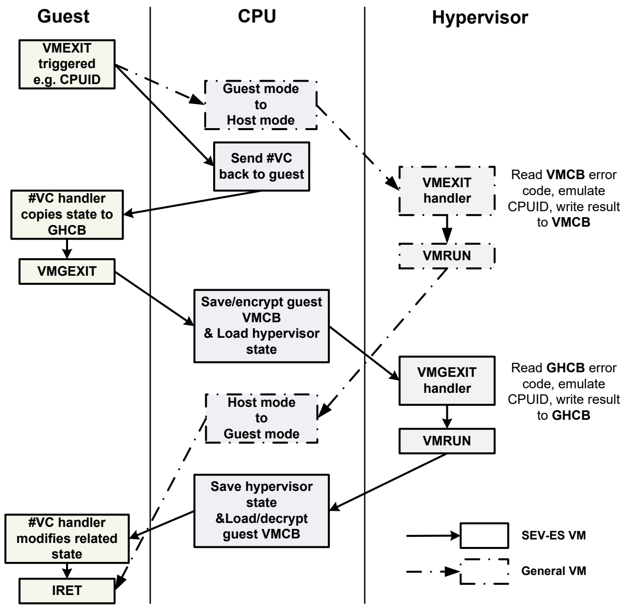
Guest
CPU
Hypervisor
VMEXIT triggered e.g. CPUID
Guest mode to Host mode
Send #VC back to guest
VMEXIT handler
Read VMCB error code, emulate CPUID, write result to VMCB
#VC handler copies state to GHCB
VMRUN
VMGEXIT
Save/encrypt guest VMCB & Load hypervisor state
VMGEXIT handler
Read GHCB error code, emulate CPUID, write result to GHCB
Host mode to Guest mode
VMRUN
Save hypervisor state &Load/decrypt guest VMCB
#VC handler modifies related state
IRET
SEV-ES VM
General VM

Fig. 2. CPUID VM Exit handling: SEV-ES CVM vs. General VM.

The hypervisor owns and manages the control area, which contains information about the events that the hypervisor must intercept, interrupt delivery information, and more. The VM save area (VMSA), which used to store the VM register state, is encrypted and integrity protected. However, accessing VM registers for VM Exit handling becomes impossible when registers are encrypted. Hence, SEV-ES allows the guest OS to choose what state information to expose to the hypervisor. This unencrypted shared memory, called the Guest-Hypervisor Communication Block (GHCB), contains only the guest register state required by the hypervisor.

SEV-ES classifies all VM Exits into two types: Automatic Exits (AE) and Non-Automatic Exits (NAE). An AE either does not require access to the guest register state (e.g., HLT instruction) or is an asynchronous event (e.g., interrupts). AEs are handled like standard VM Exits. NAEs require access to the guest register state and generate a new exception, called the "#VC exception" which has the same error code as the corresponding standard VM Exit. An NAE is handled by a "#VC handler" which runs in the guest OS, in contrast to a standard VM Exit handler which runs in the hypervisor. Figure 2 compares the execution flow in a General VM and a SEV-ES VM when a guest issues a CPUID NAE event. For a SEV-ES VM, the additional #VC handler in the guest copies relevant state to GHCB and transfers control to the hypervisor via a new VMGEXIT instruction, which triggers the real hypervisor emulation. Thus, SEV-ES requires additional load/save on GHCB and encryption/decryption on VMCB, which is a tradeoff for increased confidentiality.

C. SEV with Secure Nested Paging (SEV-SNP)

The third generation SEV-SNP [3] adds a page table called Reverse Map Table (RMP) on top of AMD-V Nested Page

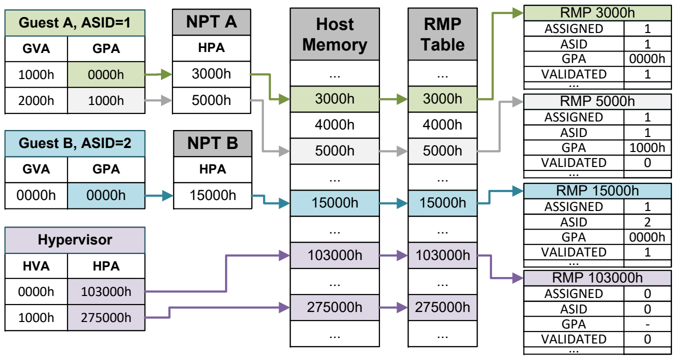

Fig. 3. Reverse Map Table in SEV-SNP

Tables (NPT) and SEV-ES. This addition is designed to mitigate security threats [8], [13]–[16] such as replay attacks, memory aliasing, memory remapping, and cache poisoning. NPT accelerates the second-level address translation of Guest Physical Address (GPA) to Host Physical Address (HPA) for each VM, that was shown in Figure 1. The RMP enforces integrity protection over second-level translation by adding an additional look-up and check after the page table walk. Figure 3 shows the memory layout when running SEV-SNP VMs. Each SEV-SNP VM has its unique NPT and ASID. The RMP table utilizes HPA as its index, and each RMP entry contains the page state of each HPA page. A page state is determined by the fields Assigned, Validated, ASID, Immutable, GPA, and VMSA [17]. The *RMP check* operation monitors the ownership of each page, which can belong to a hypervisor, a specific VM, or the AMD SP, and allows only the page owner to write to it. In Figure 3, when a user-space process (native mode) attempts to write to HPA 15000h, which belongs to Guest B, the RMP check will deny this access because the RMP entry at 15000h has an ASID of 2, exclusive to Guest B. The failure of the RMP check triggers a page fault. For an SEV-SNP VM, the RMP check is more complex. Similar to the native mode, the HVA is initially translated into an HPA, which then directly accesses the page. If Guest A tries to read page GVA 2000h, it first translates GVA to GPA 1000h and checks the NPT to translate GPA 1000h to HPA 3000h. The RMP entry 3000h is then checked as to whether the page belongs to a VM with ASID of 1, whether the current page status complies with a SEV-SNP VM definition, and importantly, whether the GPA value matches the requested page. If any of these checks fail, the VM is denied access and a page fault is triggered. In this example, the RMP check passes, so Guest A continues with its memory access.

III. Performance Evaluation of CVMs on SEV

We now explore the performance differences among CVMs on three generations of AMD SEV and General VMs.

Table I details the configuration of the testbed machines. The main experimental machine (hereafter called the *host*) is a bare-metal server, equipped with two 64-core AMD EPYC Milan 7763 processors, 256GB RAM, and a 500GB SATA SSD. In addition, one client machine with the same hardware configuration is used in network-oriented benchmarks to generate requests to the host. A successful initiation of a CVM requires the collaborative efforts of three components: the Linux kernel (with KVM features), QEMU (a VM manager process), and the guest BIOS. The host, CVMs, and client machines all run standard Ubuntu distributions with Linux kernel 5.19-rc6_v4 [18], which is maintained by AMD. The QEMU branch [19], also maintained by AMD, adds support for memory encryption for a SEV VM. SEV memory encryption is dependent on the setting of the C-bit. As the C-bit is located in the higher bit of the address, support for long-mode is necessary in the BIOS. However, SeaBIOS, which is commonly used by QEMU, doesn't support long mode. Hence AMD and TianCore have collaborated to add SEV features to the Open Virtual Machine Firmware (OVMF) BIOS [20], making it aware of all three generations of SEV CVMs. Our host supports all three generations of AMD SEV, which can be enabled as needed via QEMU configurations [21].

TABLE I
Testbed Configuration

Items	Verison
Host BIOS	DELL PowerEdge R6525 v2.11.3
Linux distribution	Ubuntu Server 22.04.2 LTS
Host & Guest Kernel	Linux 5.19-rc6_v4 (SEV dev branch) [18]
AMD SEV Firmware	1.53 build:5 for EPYC 7xx3 (Milan)
QEMU	QEMU-6.1.50-snp-v3 (SEV dev branch) [19]
OVMF BIOS	edk2-stable202302 [20]

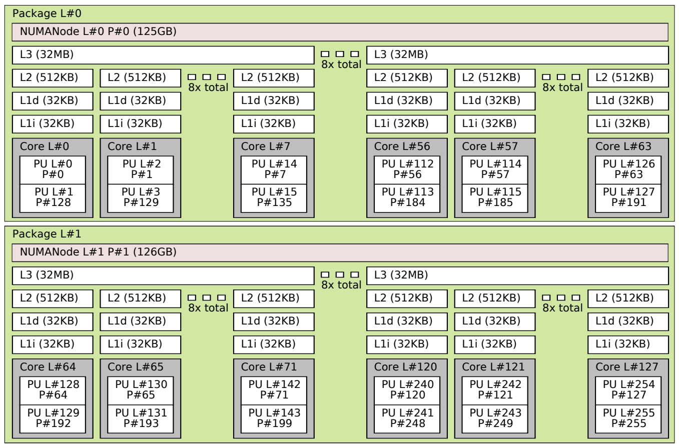

Fig. 4. AMD EPYC 7763*2 CPU topology diagram

A. NUMA Setting and VM Size

Figure 4 generated by the lstopo [22] tool, shows the CPU topology of the testbed machine. Each "Package" refers to each CPU socket. AMD EPYC 7763 has multiple dies in a multi-chip module design. Each CPU is constructed of eight Core Complex Dies (CCDs) and one I/O die. The I/O die controls all I/O operations, including its eight-channel memory controller. Each CCD contains an eight-core CPU with its own L1-L3 cache and a quick connection to the I/O die. Memory access between these 128 cores have different latencies due to Non-Uniform Memory Access (NUMA). For

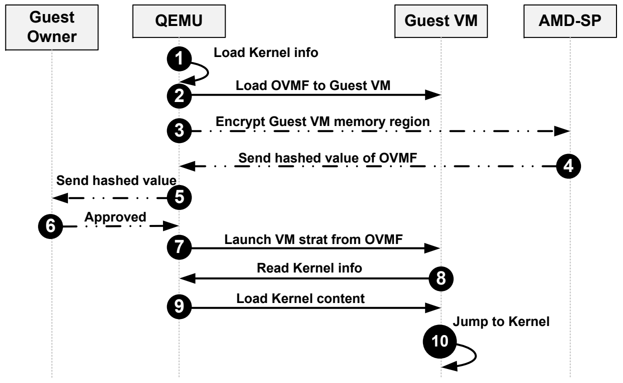

Fig. 5. SEV VM boot procedure.

instance, CPU0 (socket0, CCD0) has a NUMA distance [23] of 10 to CPU7 (same CCD as CPU0, in-node access no extra latency), 12 to CPU63 (socket0, CCD7, cross-node access: 1.2 times latency), and 32 to CPU64 (socket1, CCD0, cross-socket access: 3.2 times latency). Due to this multi-chip module design, applications on AMD EPYC should be mapped carefully to CPUs to prevent I/O or memory bottlenecks.

TABLE II
VM SIZE

VM size	Configuration
Small	2 VCPU, 4GB memory, pin on CCD8
Medium	8 VCPU, 14GB memory, pin on CCD8
Large	112 VCPU, 120GB memory, pin on CCD8-14

Table II shows three VM configurations used in our experiments – Small, Medium and Large – and their relation to the AMD EPYC 7763 CPU topology. As our goal is to observe the impact of SEV-related settings on various aspects of the VM, we strive to minimize the influence of host settings unrelated to SEV on the VM. To mitigate the aforementioned NUMA latency, we have configured Small and Medium VMs to confine the entire QEMU process within a single CCD, whereas Large VMs are confined to a single socket. The pinning processes begins at CCD8 to prevent performance disparities among CCDs. We use the “taskset” command to establish the CPU affinity for the QEMU process, thereby effectively disabling the NUMA balancing feature [24] for automatically shifting memory to nodes that frequently access it. This configuration ensures that there is no cross-node memory access for Small or Medium VMs, and no cross-socket memory access for Large VMs.

B. VM Boot Time and Host Memory Usage

TABLE III
BOOT TIME

VM size	General	SEV	SEV-ES	SEV-SNP
Small	7.79s	10.01s	14.91s	15.97s
Medium	7.76s	10.53s	14.50s	15.52s
Large	12.90s	19.76s	25.87s	30.26s

Boot time is critical for VM responsiveness, while memory usage determines the number of deployable VMs. However, the boot process for a SEV VM differs from that of a General VM, particularly in how memory regions are configured during the initialization stage. Figure 5 illustrates the SEV VM boot procedure, with steps 3, 4, 5, and 6 diverging from a General VM. After QEMU loads the OVMF BIOS to guest memory (step 2), AMD SP generates a new encryption key only for this SEV VM and encrypts the entire memory region assigned to it (step 3). Once encryption is complete, the AMD SP calculates a hash value of this encrypted memory region and sends it to the guest owner (step 4, 5). To mitigate the risk of malicious firmware injection, the guest owner, who knows the content of the given OVMF, compares the hash value. Only if the hash values matches does the guest owner authorize the VM to start (step 6), establishing trust between the VM and the guest owner. At this point, the loaded OVMF BIOS initiates the boot process (step 7). Since the OVMF BIOS is SEV aware, it executes various boot steps based on SEV settings.

Table III compares the VM boot times under different SEV settings. Boot time is defined as the interval from the issuance of the QEMU command by the host to the timestamp of a UDP packet sent from the guest OS, just after the system daemon brings all services online, as captured by the `tcpdump` utility on the host. Compared to a General VM, SEV introduces an additional boot time of less than 3 seconds for Small and Medium CVMs and around 7 seconds for Large CVMs. SEV-ES adds an additional 4 to 5 seconds compared to an SEV VM. SEV-SNP adds 1 more second to Small and Medium CVMs, and 5 seconds to Large CVMs compared to SEV-ES VMs. This leads to an increase in boot time to 15 seconds on Small and Medium CVMs and 30 seconds on a Large CVM compared to a General VM. The primary reason for the significant additional boot time for SEV-SNP is the construction of the RMP table during initialization.

We now compare the memory footprint of a SEV VM compared to a General VM. We use an identical image – an idle Ubuntu server with only the SSH service running. We measure the entire memory occupied by QEMU from /proc/PID/status, which includes the memory used by the VM. Since SEV is security oriented, a CVM’s memory regions are reserved and encrypted in advance on the host regardless of the CVM’s actual memory usage. In contrast, for a General VM, memory is allocated only when needed. For the General VM case, a Small VM occupies 1.28GB, a Medium VM occupies 1.49GB, and a Large VM occupies about 4GB. The corresponding values for the SEV VM case are 4.03GB, 14.05GB, and 120.08GB. Furthermore, memory encryption prevents the use of traditional de-duplication techniques, such as Kernel Samepage Merging (KSM) to reduce this extremely high memory usage of CVMs. Consequently, CVMs are likely to yield lower consolidation efficiency than General VMs.

C. Benchmark Performance Measurements

The benchmark tools employed were sourced either from Ubuntu’s default package sources or were compiled using gcc 11 with “-o3” option. We did not use any other compiler-level optimizations [25] [26] since our focus is on investigating

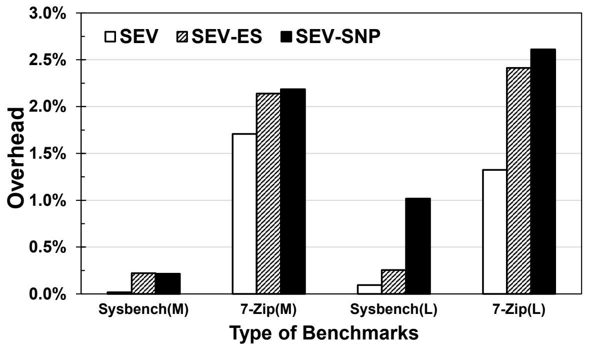

Fig. 6. Sysbench and 7-Zip compression CPU benchmark overheads for three generations of SEV compared to General VM.

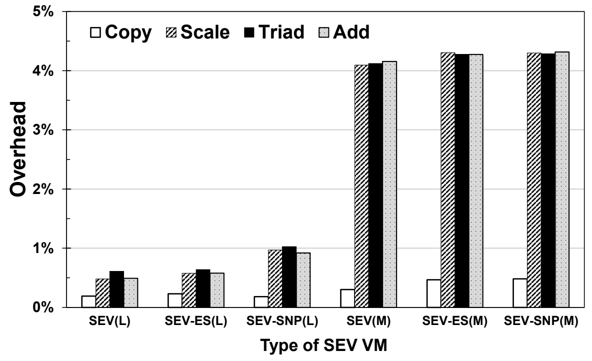

Fig. 7. Memory STREAM benchmark overheads for three SEV generations compared to General VM.

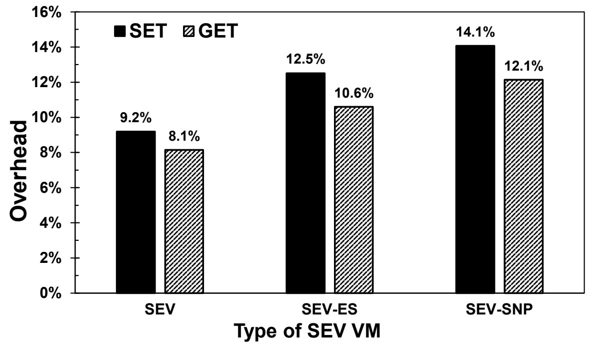

Fig. 8. Redis benchmark overhead

the general performance differences between SEV VMs and General VMs. The overhead is calculated using by first running the same benchmark tool with the exact same settings on a General VM and on a CVM to get the raw performance metrics (P_{GVM} and P_{CVM}), and then using the formula $Overhead = \frac{P_{CVM} - P_{GVM}}{P_{GVM}} * 100\%$ to calculate the overhead of the given benchmark under the same VM size. The final overhead is the average of five runs of a given benchmark.

D. CPU Performance

We used Sysbench and the 7-Zip compression benchmarks to compare the performance of CPU-intensive workloads on General and SEV VMs. Sysbench [27] is an open-source multi-threaded benchmark. When operated under a CPU workload, Sysbench performs prime number validation using the conventional method of dividing the number by all integers from 2 up to the square root of the number itself. We set the option to use the maximum available threads inside the VM and let the benchmark continue to run for 30 seconds. Figure 6 compares the results of "average executed events per second" across different SEV generations on Medium and Large VMs. Sysbench experiences less than 1% overhead in the worst case (SEV-SNP on Large VM) compared to a General VM.

7-Zip compression benchmark [28] employs the LZMA compression algorithm to continuously measure executed instructions per second. Factors such as cache latency, memory latency, and out-of-order execution hold more significance in this benchmark. The benchmark is set to use the maximum threads available within the VM, and the LZMA dictionary size remains at the default 32MB. Figure 6 shows the benchmark results for different SEV generations on Medium and Large VMs. Due to memory encryption, cache/memory access latency increases, which causes this benchmark to reflect an overhead of up to 1.5% on SEV, and up to 2.5% on SEV-ES and SEV-SNP. Hence, while a SEV VM does not experience significant CPU overheads compared to a General VM, there is a small discernible impact on workloads sensitive to cache/memory latency.

E. Memory performance

Since SEV fundamentally aims at protecting data during processing, it sits in the critical performance path for memory-intensive workloads. We use two memory-intensive benchmarks – STREAM and Redis – to ascertain the difference in performance between a General VM and a SEV VM.

The STREAM [29] benchmark is a synthetic benchmark to measure the sustainable main memory bandwidth for simple vector kernels. Each vector operation presents a different cache/memory access pattern. We configured OpenMP [30] to enable parallelized execution of STREAM and used the default setting of the problem size. Figure 7 compares the overhead of SEV VMs to General VMs. For a Medium VM, all three types of SEV settings bring a similar overhead of up to 4.3% on Scale, Triad, and Add sub-benchmarks. For a Large VM, the performance overhead reduces because the overall performance of a Large VM is about three times that of a Medium VM. Thus, the performance loss appears smaller due to a larger performance base.

Redis [31] is an in-memory, key-value data store that can be used as a database, cache, message broker, and queue. Redis Benchmark is a Redis built-in tool that measures the performance of a given Redis server by emulating multiple database clients that continuously send commands to the server. We ran the Redis clients on another machine connected to the host machine over a 10Gbps network link. We measured "GET" and "SET" commands representing memory write and read operations. We used 8-byte data size of each "SET/GET" request. The total of 50 million "GET" requests and 20 million "SET" requests were used with the pipeline set to

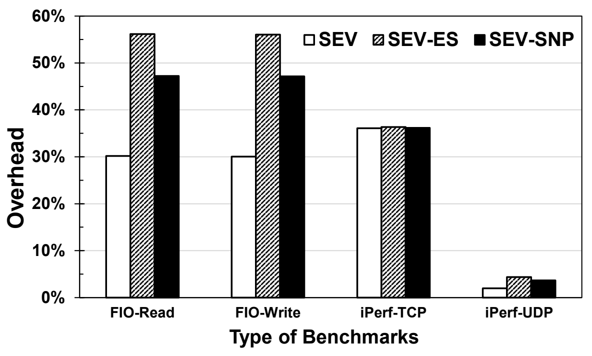

Fig. 9. FIO & iPerf I/O benchmark overheads

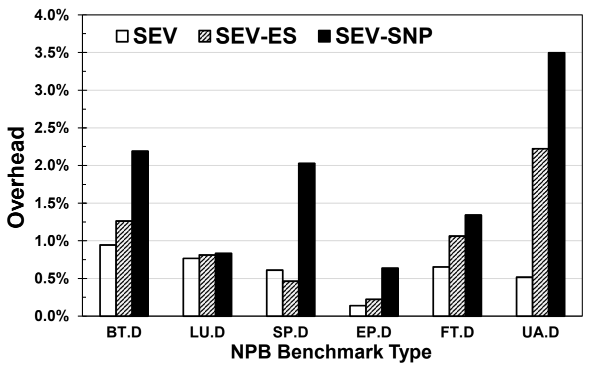

Fig. 10. NPB benchmark overhead

24. The Redis server inside the VM also enabled multi-threading for better overall performance. Redis Benchmark is reports performance as "executed requests per second." Figure 8 shows the Redis Benchmark overhead on a Small VM. SEV alone resulted in about a 9% and 8% performance loss for write and read operations, respectively. SEV-ES and SEV-SNP further exacerbated the performance drop, leading to up to a 14% loss on write operations and a 12% loss on read operations. Since the Redis benchmark generates a combination of memory and network I/O workloads, we see a correspondingly larger performance overhead compared to the STREAM benchmark, which is purely memory-intensive. Next, we examine the I/O overheads of CVM in more detail.

F. I/O Performance

We measured the disk and network throughput using Flexible I/O tester (FIO) [32] and iPerf [33] benchmarks, respectively. The virtual disk is an unencrypted, uncompressed disk in qcow2 format. While an encrypted disk would ordinarily be utilized to enhance a CVM's security, we chose an unencrypted disk to examine the performance impact of a pure SEV setup. Currently, SEV VM's I/O relies on IOMMU DMA and employs a shared memory segment between the VM and the hypervisor, known as the Software I/O Translation Buffer (SWIOTLB). For optimal performance, we used para-virtualized virtio-scsi-pci and virtio-net-pci as QEMU back-ends with IOMMU DMA enabled. The size of the SWIOTLB was set to 256MB for Small CVM and 1024MB for Medium CVMs, to avoid unexpected VM crashes due to a lack of available SWIOTLB under I/O-intensive workloads.

FIO reports disk I/O Operations per second (IOPS). We set an I/O depth of 128, block size of 4KB, non-buffered I/O, a mixed workload comprising 75% random read and 25% random write, and let the benchmark continue to run for 30 seconds. For a General Medium VM, FIO reports read throughput of 143.6K IOPS and write throughput of 47.9K IOPS. Using these General VM values as baseline, Figure 9 shows the FIO benchmark overhead of different SEV generations on a Medium CVM. SEV-ES demonstrates the worst performance of all tested setups, with a 56% loss in disk read and write performance, primarily due to the addition of #VC handler and GHCB communication.

iPerf measures the maximum achievable bandwidth via TCP or UDP connections from a server to a client. iPerf server ran inside either a General VM or a CVM, and the iPerf client ran on another machine linked with a 10Gbps connection to the host server machine. The throughput reported by iPerf is the same across different SEV settings and the General VM, with TCP having 9.4 Gbps and UDP having 2.3 Gbps. As the 10Gbps physical bandwidth limit became the bottleneck of this experiment, we turned to use the local loop-back interface on the host side. We created a Linux bridge device for local switching, and a TAP device connects the bridge device. We allocated a dedicated subnet containing only the host local bridge and VM. This configuration increased the throughput of iPerf for the General VM to 65.13 Gbps for TCP and 5.36 Gbps for UDP. Using these values for General VM as the baseline, Figure 9 shows the iPerf overhead from a Medium VM to the Host through the host loop-back network for different SEV settings. All three SEV settings exhibit about a 36% degradation in TCP throughput, primarily due to the SWIOTLB implementation mentioned earlier. UDP throughput, on the other hand, is mainly limited by the network stack and single-core CPU performance, with SEV settings causing only a 3%-4% drop. The performance loss in I/O is considerable and largely attributable to the current design limitations of SEV that force I/O operations to rely on different forms of shared memory between the CVM and hypervisor.

G. High Performance Computing (HPC) Workload

The NAS Parallel Benchmarks (NPB) [34] is a collection of benchmarks designed to measure the performance of parallel supercomputers, reflecting realistic and complex computing scenarios. Similar to STREAM, we utilized OpenMP to establish the parallelized benchmark environment within a VM. These benchmarks primarily originate from the application of Computational Fluid Dynamics. Of the nineteen benchmarks in total, we picked six benchmarks related to pure computation, data movement, and parallel I/O-related benchmark. Figure 10 shows the performance overhead of SEV VMs compared to a General VM. We choose the predefined level D as the problem size for each benchmark, representing large test problems. Each benchmark takes approximately 20 to

45 minutes to complete in our Large VM. The performance impact from SEV varies based on the different workloads. However, SEV-SNP brings a greater performance impact compared to SEV and SEV-ES on all benchmarks due to its additional RMP check operation on every memory access. The UA benchmark, which relies on parallel I/O, reveals that SEV-ES and SEV-SNP have a significantly adverse impact on I/O-intensive workloads, primarily due to the design of SWIOTLB. It is important to note that the NPB Benchmark exhibits a large difference when NUMA balancing is enabled without pinning the VCPU, and we have observed up to more then 50% overhead on some benchmarks across different SEV settings compared to a General VM. This substantial difference is the primary reason we decided to set VCPU affinity, disable NUMA balancing, and match the VM to the host CPU topology.

H. Other Limitation of SEV

Despite SEV's important functionality of adding confidentiality to VMs, it also reduces some of the capabilities of a VM. For instance, a system can only support a certain number of SEV VMs simultaneously – 512 on our testbed EPYC 7003 platform. PCI pass-through, a form of direct I/O access for VMs, is also more complicated to support, if not impossible, at present. Furthermore, the most significant drawback is the lack of VM checkpointing and live migration support. Checkpointing and live migration are relatively straightforward with a General VM, mainly because the hypervisor maintains full control of the VM. Currently, two solutions are being explored to address this issue for CVMs. The first solution is an expensive approach [35] that allows the hypervisor to send requests to decrypt and re-encrypt pages one by one using a newly generated transfer key specifically for the checkpointing or live migration operation. This approach heavily relies on the AES engine's throughput. The second solution [36] involves adding helper threads inside the CVM, but hidden from the Guest OS, to assist in setting up the security transition tunnel and tracking dirty memory by setting pages as read-only. This approach reduces the encryption/decryption overhead as the helper threads can directly read the plain-text content of a page since they operate inside the CVM. At the time of writing this paper, we were unable to successfully reproduce and evaluate any of the above publicly available live migration/checkpointing codes for CVMs, but we look forward to doing so once stable prototypes are made available.

IV. Related Work

Prior research efforts on performance analysis of AMD SEV have been restricted to either the first generation SEV or only to specific types of workloads. Our work complements them to provide a comprehensive and more up-to-date understanding of this important and rapidly evolving area.

Akram et. al. [5] first compared the performance of scientific workloads between first generation SEV VMs and SGX [37]. They attributed the NPB performance overhead from an SEV VM mainly to the host side's default NUMA memory allocation policy. We reached a similar conclusion in our tests. Additionally, we tried to eliminate the memory allocation problem by disabling NUMA balancing, setting a reasonable size for the VM, and pinning VCPUs. We found that the performance degradation on the NPB benchmark still exists.

Göttel et. al. [6] compared the memory bandwidth, latency, and system energy cost of the first generation SEV and SGX using a variety of micro and macro benchmarks, including Redis server LOAD and SCAN operations. We used Redis server SET and GET operations as one of our benchmarks on memory, finding higher overheads compared to their results in all three generations of SEV VMs. Mofrad et.al. [7] compared SGX enclave and first generation SEV on features, security level, and CPU performance overhead. They used quicksort, MD5, and floating-point workloads to measure the pure computation overhead. Similar to our results, they found that SEV did not introduce much overhead on CPU performance. Li et. al. [8] discussed the performance overheads and security issues in first generation SEV VMs. In this paper, we were able to reproduce the disk and network I/O overhead results with different benchmark tools and extend the discussion to all three generations of SEV implementation. Bifrost [9] is a para-virtualized I/O design that aims to eliminate the network I/O overhead in SEV-ES VMs. The authors found that inefficiencies in bounce buffer processing as well as VM Exit events cause the CVM's VCPUs to lose more than 50% of CPU cycles under I/O-intensive workloads. They proposed several optimizations to improve CVM network I/O performance significantly, even surpassing a General VM.

Some efforts have also demonstrated new functionalities using AMD SEV. Brasser et. al. [38] implemented a prototype of a confidential container using first generation SEV. They focused on container use cases with Nginx, Apache, and Redis performance. Pecholt et. al. [39] implemented a confidential container prototype with live migration support. It is based on the first-generation SEV technology and is modeled on the first approach mentioned in Section III-H, incorporating the necessary remote authentication and encrypted data transfer. As per reported numbers, it took more than 45 minutes to migrate a 4GB container due to performance limitations of the AMD SP, which is extremely slow compared to live migration of a General VM of similar size. vSGX [40] emulates an Intel SGX enclave on top of the AMD SEV-ES platform to demonstrate the flexibility of the whole VM encryption approach to support other runtimes. More use cases are likely to emerge as the technology matures.

AMD is actively engaged in endeavors to alleviate the performance degradation associated with SEV. In March 2023, they proposed a new DMA design called Trusted I/O for Secure Encrypted Virtualization (SEV-TIO) [41], to rectify the inherent flaws of the original design. Concurrently, they are developing the Secure VM Service Module (SVSM) [42], a dedicated guest communication interface specifically tailored for SEV-SNP VM guests, providing enhanced security control and greater adaptability within a CVM.

V. Conclusion

CVM fills a critical gap in confidential computing by providing transparent hardware support for encrypted VMs. Three generations of AMD SEV have progressively addressed this problem by adding more hardware abstraction layers to provide memory encryption, register state encryption, and page table integrity protection. However, as shown in this paper, these new abstraction layers have emerged as new performance bottlenecks, especially for I/O and memory-intensive workloads. Despite these overheads, CVMs continue to draw strong interest from industry and academia due to the very tangible problem that they solve. Hence, addressing performance issues in CVMs will remain an important priority in the future.

Acknowledgment

Results in this paper were obtained using the Chameleon testbed [43] supported by National Science Foundation, USA. We gratefully acknowledge the support of Chameleon testbed staff with our numerous reconfiguration requests.

References

[1] D. Kaplan, J. Powell, and T. Woller, "AMD Memory Encryption," https://www.amd.com/system/files/TechDocs/memory-encryption-white-paper.pdf, 2021.

[2] D. Kaplan, "Protecting VM Register State With SEV-ES," https://www.amd.com/system/files/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf, 2017.

[3] AMD, "AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More," https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020.

[4] Intel, "Intel® Trust Domain Extensions," https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf, 2023.

[5] A. Akram, A. Giannakou, V. Akella, J. Lowe-Power, and S. Peisert, "Performance analysis of scientific computing workloads on general purpose tees," in *Proc. of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021.

[6] C. Göttel, R. Pires, I. Rocha, S. Vaucher, P. Felber, M. Pasin, and V. Schiavoni, "Security, performance and energy trade-offs of hardware-assisted memory protection mechanisms," in *Proc. of IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2018.

[7] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of Intel SGX and AMD memory encryption technology," in *Intl. Workshop on Hardware and Architectural Support for Security and Privacy*, 2018.

[8] M. Li, Y. Zhang, and Z. Lin, "Exploiting unprotected I/O operations in AMD's secure encrypted virtualization," in *Proc. of USENIX Security*, 2019.

[9] D. Li, Z. Mi, C. Ji, Y. Tan, B. Zang, H. Guan, and H. Chen, "Analysis and optimization of network I/O tax in confidential virtual machines," in *Proc. of USENIX Annual Technical Conference (ATC)*, 2023.

[10] AMD, "AMD64 Architecture Programmer's Manual Volume 2: System Programming," https://www.amd.com/system/files/TechDocs/24593.pdf, 2023.

[11] Wikipedia, "AMD Virtualization (AMD-V)," https://en.wikipedia.org/wiki/X86_virtualization#AMD_virtualization_(AMD-V), 2022.

[12] ARM Limited, "TrustZone security extensions," https://developer.arm.com/documentation/ddi0301/h/introduction/trustzone-security-extensions, 2023.

[13] AMD Security Bulletin AMD-SB-1023, "TLB Poisoning Attacks on AMD Secure Encrypted Virtualization (SEV)," https://www.amd.com/en/resources/product-security/bulletin/amd-sb-1023.html.

[14] L. Wilke, J. Wichelmann, F. Sieck, and T. Eisenbarth, "undeSErVed trust: Exploiting permutation-agnostic remote attestation," in *Proc. of IEEE Security and Privacy Workshops (SPW)*, 2021.

[15] M. Morbitzer, S. Proskurin, M. Radev, M. Dorfhuber, and E. Q. Salas, "SEVerity: code injection attacks against encrypted virtual machines," in *Proc. of IEEE Security and Privacy Workshops (SPW)*, 2021.

[16] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, "SEVered: Subverting AMD's virtual machine encryption," in *Proc. of European Workshop on System Security*, 2018.

[17] *SEV Secure Nested Paging Firmware ABI Specification*, https://www.amd.com/system/files/TechDocs/56860.pdf, AMD Std., Rev. 1.54, 2022.

[18] AMDESE, "Linux Kernel: sev-snp-iommu-avic_5.19-rc6_v4," https://github.com/AMDESE/linux/tree/sev-snp-iommu-avic_5.19-rc6_v4, 2022.

[19] ——, "QEMU-6.1.50-snp-v3," https://github.com/AMDESE/qemu/tree/snp-v3, 2022.

[20] TianoCore, "edk2-stable202302," https://github.com/tianocore/edk2/releases/tag/edk2-stable202302, 2023.

[21] AMD, "Using SEV with AMD EPYC™ Processors," https://www.amd.com/content/dam/amd/en/documents/developer/58207-using-sev-with-amd-epyc-processors.pdf, 2023.

[22] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in HPC applications," in *Euromicro Intl. Conf. on Parallel, Distributed and Network-Based Processing*, 2010.

[23] *Advanced Configuration and Power Interface (ACPI) Specification*, https://uefi.org/htmlspecs/ACPI_Spec_6_4_html/index.html, Unified EFI Forum, Inc Std., Rev. 6.4, 2021.

[24] R. van Riel and V. Chegu, "Automatic NUMA balancing," *Red Hat Summit*, 2014.

[25] AMD, "AMD Optimized C/C++ Compiler (AOCC)," https://www.amd.com/en/developer/spack/spac-aocc.html, 2023.

[26] ——, "AMD optimized CPU libraries (AOCL)," https://www.amd.com/en/developer/spack/spac-aocl.html, 2023.

[27] Alexey Kopytov, "sysbench," https://github.com/akopytov/sysbench, 2023.

[28] I. Pavlov, "7-Zip LZMA Benchmark," https://www.7-cpu.com/, 2023.

[29] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *TCCA Newsletter, 19-25*, Dec. 1995.

[30] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *Comput. Sc. and Eng. 5(1), 46-55*, 1998.

[31] Redis Ltd., "Redis benchmark," https://redis.io/docs/management/optimization/benchmarks/, 2023.

[32] J. Axboe, "fio - Flexible I/O tester rev. 3.35," https://fio.readthedocs.io/en/latest/fio_doc.html, 2017.

[33] "iPerf: The TCP/UDP bandwidth measurement tool," http://dast.nlanr.net/Projects/Iperf/.

[34] D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, A. Woo, and M. Yarrow, "The NAS parallel benchmarks 2.0," Technical Report NAS-95-020, NASA Ames Research Center, Tech. Rep., 1995.

[35] A. Kalra, "Add SEV guest live migration support," https://lore.kernel.org/qemu-devel/cover.1628076205.git.ashish.kalra@amd.com/, 2021.

[36] T. Feldman-Fitzthum and D. Murik, "Secure Live Migration of Encrypted VMs," https://research.ibm.com/publications/secure-live-migration-of-encrypted-vms, 2021.

[37] S. Johnson, R. Makaram, A. Santoni, and V. Scarlata, "Supporting intel sgx on multi-socket platforms," https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/supporting-intel-sgx-on-mulit-socket-platforms.pdf, Retrieved Jul 2023.

[38] F. Brasser, P. Jauernig, F. Pustelnik, A.-R. Sadeghi, and E. Stapf, "Trusted container extensions for container-based confidential computing," *arXiv preprint arXiv:2205.05747*, 2022.

[39] J. Pecholt, M. Huber, and S. Wessel, "Live migration of operating system containers in encrypted virtual machines," in *CCS Workshop*, 2021.

[40] S. Zhao, M. Li, Y. Zhangyz, and Z. Lin, "vSGX: Virtualizing SGX enclaves on AMD SEV," in *Proc. of IEEE Symposium on Security and Privacy*, 2022.

[41] AMD, "AMD SEV-TIO: Trusted I/O for Secure Encrypted Virtualization," https://www.amd.com/system/files/documents/sev-tio-whitepaper.pdf, 2023.

[42] *Secure VM Service Module draft specification*, https://www.amd.com/system/files/TechDocs/58019-svsm-draft-specification.pdf, AMD Std., Rev. 0.50, 2022.

[43] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, "Lessons learned from the Chameleon testbed," in *Proc. of USENIX Annual Technical Conference (ATC)*, July 2020.